Computer Science Department, Ben-Gurion University of the Negev

Mini-project in Scientific Computing - Spring 2023

Mini-project: Deep Learning from Scratch

Due on: part 1: February 15th. part 2: April 7th.

1 Project Description

The goal of this project is to gain an idea of how deep learning frameworks work behind the scenes (the back-prop, optimization, influence of hyper parameters), and to gain experience in scientific programming and stochastic optimization methods. This will help you switch between the available deep learning frameworks, and understand how to extend standard libraries when certain modules you need do not exist. A secondary (yet equally as important) goal of this project is to provide the methodologies and gain experience in the development of complex optimization frameworks with a lot of moving parts.

In this project we will implement a simple neural network for classification of small vectors. We will be given with "training" labeled data samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ and another set of "validation" data $\{(\mathbf{x}_i^v, y_i^v)\}_{i=1}^{m_v}$. The goal is to define and train a neural network that knows how to classify the data samples (given some sample $\mathbf{x}$, predict its label y).

You will define some structure of a network (called architecture) which has unknown weights that receives a point and outputs a distribution of probabilities for each label. Then, you will train the network to classify the data samples by optimizing an objective function that tries to fit the predictions with the given labels.

Our network will have the following architecture:

$$L(\{\theta_l\}_{l=1}^{L}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\theta^{(L)}, \mathbf{x}_i^{(L)}, y_i) \tag{1}$$

$$s.t \qquad \mathbf{x}_i^{(l+1)} = f(\theta^{(l)}, \mathbf{x}_i^{(l)}); \quad l = 1, ..., L-1 \tag{2}$$

where $\ell()$ is the softmax objective function defined in the lectures, and $\mathbf{x}_i^{(1)}$ denotes the given data sample i. The architecture be either a standard Neural Network

$$f(\theta^{(l)}, \mathbf{x}_i^{(l)}) = \sigma(\mathbf{W}^{(l)}\mathbf{x}_i^{(l)} + \mathbf{b}^{(l)});$$

or a simple ResNet (you may remove the activation at the end of this block and ignore $\mathbf{b}_2^{(l)}$ as in the notes.)

$$f(\theta^{(l)}, \mathbf{x}_i^{(l)}) = \sigma(\mathbf{x}^{(l)} + \mathbf{W}_2^{(l)}\sigma(\mathbf{W}_1^{(l)}\mathbf{x}^{(l)} + \mathbf{b}_1^{(l)}) + \mathbf{b}_2^{(l)})$$

We will denote by $\theta^{(l)}$ the set of weights of layer l that we need to learn. For the last layer L we have to learn the softmax weights:

$$\theta^{(L)} = \left\{ \{\mathbf{w}_j^{(L)}\}_{j=1}^{n_{labels}} \in \mathbb{R}^n, \text{ and the bias vector } \mathbf{b}^{(L)} \in \mathbb{R}^{n_{labels}}. \right\}$$

For each data sample $\mathbf{x}_i$, the predicted label is the label with highest probability in the softmax function.

For the standard layers we have

$$\theta^{(l)} = \{\mathbf{W}^{(l)} \in \mathbb{R}^{n_2 \times n_1}, \mathbf{b}^{(l)} \in \mathbb{R}^{n_2}\},$$

where n_1 and n_2 may differ. In particular, it is common (and necessary) to enlarge the width of the network (the size of the vectors $\mathbf{x}_i^{(l)}$) as the layers progress. For the residual layers $l = L-1, ..., 1$ we have

$$\theta^{(l)} = \{\mathbf{W}_1^{(l)} \in \mathbb{R}^{n_2 \times n_1}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{n_1 \times n_2}, \mathbf{b}_1^{(l)} \in \mathbb{R}^{n_2}, \mathbf{b}_2^{(l)} \in \mathbb{R}^{n_1}\},$$

which needs to be learned.

2 Specific Tasks

This project is large and has many moving parts. Below are the steps that you need follow on your way to the big goal: a trained network that can classify in high accuracy. All the steps should be submitted in a report, and you get part of the grade for each of them.

2.1 Part I: the classifier and optimizer (40%)

1. Write the code for computing the loss function "soft-max regression" and its gradient with respect to $\mathbf{w}_j$ and the biases. Make sure that the derivatives are correct using the gradient test (See the subsection "Gradient and Jacobian Verification" in the notes). You should demonstrate and submit the results of the gradient test.

2. Write the code for minimizing an objective function using SGD or some other SGD variant (SGD with momentum, for example). Demonstrate and verify that your optimizer works on a small least squares example (add plots and submit the code itself).

3. Demonstrate the minimization of the softmax function using your SGD variant. Plot a graph of the success percentages of the data classification after each epoch—for both the training data and the validation data. For the plots, you may only use a random subsample of the two data sets, instead of each time computing the loss for the whole data, as these are extra computations that are not related to the optimization itself. Try a few learning rates and min-batch sizes and see how they influence the performance. Submit the graphs only for your best option, but also write about your tries. Run as many iterations of SGD as you see that is needed (i.e., stop when more iterations do not improve the accuracy, even if the learning rate decreases).

2.2 Part II: the neural network (60%)

1. Write the code for the standard neural network. Including the forward pass and backward pass (the computation of the "Jacobian transpose times vector"). See that the Jacobian tests work and submit the tests (using plots, as demonstrated in the course notes). **This part should not be overlooked - you should show the Jacobian tests for all layers**.
   Remark 1: for the Jacobian tests use the tanh() activation function, as it is differentiable and will behave properly in the gradient tests. The ReLU function is piecewise linear and non-smooth and may lead to weird-looking gradient tests. After the gradient test passes, you may use either activation functions in the network.
   Remark 2: In the network you only use the "Jacobian transposed", but the Jacobian test as is goes through the Jacobian itself, which you need to program only for the test. A way to by-pass this is though " The direct Jacobian transposed test" – see the notes. You may use either one of the approaches.

2. Repeat the previous section for the residual neural network (use any variant you want, as long it has a skip connection).

3. Now, after we've verified that all the individual parts are OK, we will examine that their combination together in the whole network also works—the forward pass and the backward pass of the network with L layers (L is a parameter). See that the gradient of the whole network (softmax + layers) passes the gradient test. **Submit this verification**.
   In terms of implementation, you may define the weights of the network (the union of the parameters for all the layers) as a list of arrays (different array for each layer) or as one long vector that you slice along the way. Both ways can be used for the perturbation vector that you use in the grad test – you just need to define the inner product correctly. Also - note that since the network ends with the loss (a scalar function) this is a gradient test and not a Jecobian test.

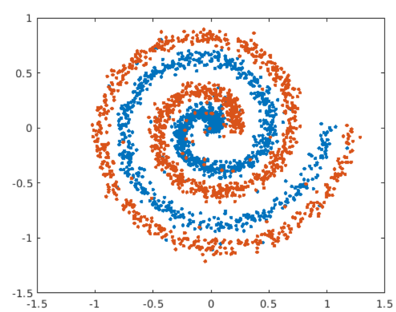
4. Repeat section 2.1.3 for the entire network. Try a few network lengths and see how this influences the performance (write the details of your networks, hyperparameters and experiments). Write your conclusions and demonstrate them.

5. NNs are expensive, and there is an effort in the literature to reduce the costs of NN. One way is to define NNs with less parameters and see how much accuracy we can squeeze. Repeat the previous section, only now use only $100C$ scalar parameters for your network, where C is the number of classes (depend on the data set). What strategy yields the best performance to you? Write the best accuracies that you got and explain the parameter count of the network.

3 Data sets

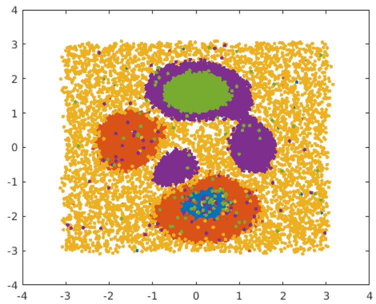
Run your experiments in the sections above on one of the first two data sets where the dimension $n = 2$ (either option 1 or 2) and on the the data set where $n = 5$ (option 3). For each data set mat file, the matrices `Yt,Ct` represent the training data and `Yv,Cv` represent the validation data.

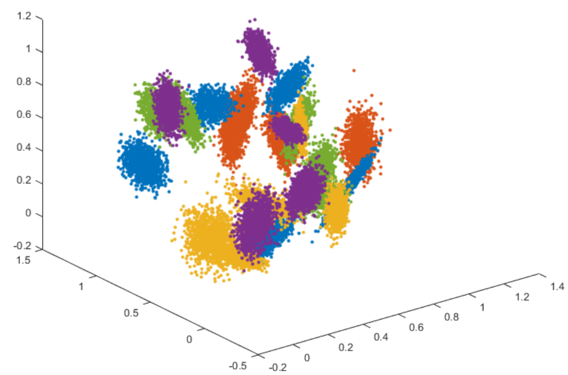
1. A two dimensional data "SwissRoll" with two classes (colored red and blue).



2. A two dimensional data "Peaks" with 5 classes (five colors).

3. A five dimensional data "GMM" data with 5 classes (five colors). For illustration, the figure below has $n = 3$, but in the actual data $n = 5$.


4
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4


1.2
1
0.8
0.6
0.4
0.2
0
-0.2
1.5
1
0.5
0
-0.5
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4